TERMINATION ANALYSIS OF SAFETY VERIFICATION FOR NON-LINEAR ROBUST HYBRID SYSTEMS

Zhikun She
LMIB and School of Mathematics and Systems Science, Beihang University, 100191 Beijing, China

Keywords: Robust hybrid systems, Abstraction refinement, Reachability constraints, Termination analysis.

Abstract: Safety verification of hybrid systems is in general undecidable. Due to practical applications, it is sufficient to only consider robustly safe hybrid systems in which a slight perturbation is guaranteed to result in the same desired safety property. In this paper, we provide a constraint based abstraction refinement for safety verification of nonlinear hybrid systems and prove that this refinement procedure will terminate for robustly safe nonlinear hybrid systems.

1 INTRODUCTION

Hybrid systems (Alur et al., 1995; Schaft and Schumacher, 2000; Ratschan and She, 2007) is a class of dynamical systems, which in addition to the discrete events also contain continuous behaviors that evolve according to differential equations or difference equations. Many examples of hybrid systems (Fehnker and Ivančić, 2004) are obtained when a digital system is embedded in an analog environment which, in many cases, is described by physical laws that are formulated using differential equations or difference equations. Such systems usually operate in safety-critical domains, for example, inside automobiles, aircrafts, and chemical plants. Thus, an important task is to verify that a given hybrid system is safe, that is, to verify that every trajectory of a given hybrid system starting from an initial state never reaches an unsafe state (i.e., a so-called “bad” state).

The safety verification problem of hybrid systems is in general undecidable (Henzinger et al., 1998) and terminating algorithms exist only for certain special cases, for example, linear hybrid automata (Henzinger et al., 1998) and o-minimal hybrid automata (Lafferriere et al., 1999).

Since hybrid systems often model a given real system in practice with perturbations, the notation of robustness (Henzinger and Raskin, 2000; Fränzle, 2001; Girard and Pappas, 2006; Damm et al., 2007; Julius et al., 2007) has been introduced to model the given real system up to perturbations. Hence, from the practical viewpoint, it is sufficient to only consider robust systems in which a slight (quantifiable) perturbation is guaranteed to result in the same desired qualitative properties (e.g., safety and stability).

In this paper, we will provide a constraint based approach for safety verification of continuous-time hybrid systems (Ratschan and She, 2007; Frehse, 2008) such that the termination of our approach is guaranteed even for a very rich class of models, which involve function symbols in $\{+, \times, \hat{}, \sin, \cos, \exp\}$. Note that unless otherwise specified, hybrid systems in this paper denote continuous-time hybrid systems.

Following our earlier works (Ratschan and She, 2007; Ratschan and She, 2006; She and Zheng, 2008), we continue to use constraints for describing hybrid systems. In addition, for describing robust hybrid systems, we use the solution sets to the corresponding constraints defined for hybrid systems with small perturbations.

For verifying safety property of hybrid systems, we use an abstraction refinement technology. That it, for a concrete hybrid system, we first split its state space into boxes and then abstract it to a finite transition system which over-approximates the concrete system in a conservative way. During the refinement procedure, we also include more information from the concrete system into the abstract one, which is done by constructing a reachability constraint, checking whether a certain state fulfills this constraint and removing states that do not fulfill this constraint by an interval based pruning algorithm. However, the interval based abstraction refinement in some cases results in the wrapping effect (Neumaier, 1993), which will be explained in Subsection 3.1. For reducing such a wrapping effect, we propose a quantifier elimination based remedy. That is, we first construct a constraint to describe the reachable set on the boundaries of boxes such that every free variable only occurs once; then, we employ a special quantifier elimination method to get the exact solution set to this constructed constraint; finally, we use this exact solution set in the reachability constraint for further computation.

Moreover, based on our proposed remedy, we can prove that our abstraction refinement procedure will eventually terminate for robustly safe hybrid systems.

Compared to the discrete time model in (Damm et al., 2007), there are variables for describing differentiation, which do not vary over the state space and may take unbound values. Moreover, compared to the counter-example guided abstraction refinement (CEGAR) based approach (Klaedtke et al., 2007), we avoid solving a large reachability constraint formulating states reachable via a trajectory over a finite number of abstract states (i.e., boxes).

This paper is organized as follows. In Section 2 we formulate our basic notions on hybrid systems and robust hybrid systems. In Section 3, we introduce a constraint based abstraction refinement for safety verification of hybrid systems, associated with a remedy for reducing the wrapping effect in Subsection 3.1 and a special quantifier elimination method in Subsection 3.2. In Section 4 we analyze the termination of our abstraction refinement procedure with our proposed remedy for robustly safe hybrid systems. In Section 5 we conclude the paper.

2 ROBUST HYBRID SYSTEMS

We fix a variable m ranging over a finite set of discrete modes $\mathtt{M} = \{m_1, \ldots, m_n\}$ and variables $x_1, \ldots, x_k$ ranging over closed real intervals $I_1, \ldots, I_k$. We denote by S the resulting state space $\mathtt{M} \times I_1 \times \cdots \times I_k$ and let $X = \{x_1, \ldots, x_k\}$. For denoting the derivatives of $x_1, \ldots, x_k$ we use variables $\dot{x}_1, \ldots, \dot{x}_k$, ranging over $\mathbb{R}$ each, and let $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_k\}$. Moreover, for denoting the targets of jumps, we use variables $m', x'_1, \ldots, x'_k$ ranging over $\mathtt{M}$ and $I_1, \ldots, I_k$ and let $X' = \{x'_1, \ldots, x'_k\}$. For simplicity, we sometimes use the vector $\vec{x}$ to denote $x_1, \ldots, x_k$, and $(m, \vec{x})$ to denote a state. Similar notations are used for $\vec{x'}$ and $\vec{\dot{x}}$.

In order to describe hybrid systems we use constraints that are arbitrary Boolean combinations of equalities and inequalities over terms. These constraints are used, on the one hand, to describe the possible flows and jumps and, on the other hand, to mark certain parts of the state space (e.g., the set of initial/unsafety states).

Definition 1.

1. *An arithmetic expression is a term (in the predicate-logical sense) with function symbols in* $\{+, \times, \hat{}\,, \sin, \cos, \exp\}$.
2. *An atomic arithmetic state space constraint is of form $e\,r\,c$, where e is an arithmetic expression, $r \in \{=, <, >, \leq, \geq\}$ is a relation operator, and c is a real-valued constant.*
3. *A mode constraint is an expression of from $m = m_i$ or $m' = m_i$, where $m_i \in \mathtt{M}$.*
4. *A state space constraint is a Boolean combination of forms $ms \rightarrow aas$, where ms is a mode constraint containing only m, $\rightarrow$ is a Boolean implication, and aas is a Boolean combination of atomic arithmetic state space constraints containing variables only in X.*
5. *A flow constraint is a Boolean combination of forms $ms \rightarrow fs$, where ms is a mode constraint containing only m, $\rightarrow$ is a Boolean implication, and fs is a Boolean combination of atomic arithmetic state space constraints containing variables only in $X \cup \dot{X}$.*
6. *A jump constraint is a Boolean combination of forms $js \rightarrow js'$, where js is a state space constraint, $\rightarrow$ is a Boolean implication, and js' is a Boolean combination of mode constraints containing only m' and atomic arithmetic state space constraints containing variables only in $X \cup X'$.*

Definition 2. *A* hybrid system *over the state space S is a tuple $(Flow, Jump, Init, UnSafe)$ consisting of a flow constraint $Flow$ describing the continuous dynamical evolutions, a jump constraint $Jump$ describing the set of possible discrete jumps, a state space constraint $Init$ describing the set of initial states, and a state space constraint $UnSafe$ describing the set of unsafe states.*

For simplicity, we use *Flow* for describing both the flow constraint and the subset of $S \times \mathbb{R}^k$ satisfying this flow constraint. Similar conventions are also used for *Jump*, *Init* and *UnSafe*. Thus, a hybrid system $\mathcal{H}$ can also be formulated as a tuple $(Flow, Jump, Init, UnSafe)$, where $Flow \subseteq S \times \mathbb{R}^k$, $Jump \subseteq S \times S$, $Init \subseteq S$, and $UnSafe \subseteq S$.

Definition 3.

1. *A* flow *of length l in a mode m is a function $r : [0, l] \mapsto S$ such that*
   - $(r(t), \dot{r}(t)) \in Flow$*, where $\dot{r}(t)$ denotes the derivative of the projection of r to its continuous part, and*
   - *for all $t \in [0, l]$, the mode of $r(t)$ is m.*

For simplicity, for a flow r, we will use $len(r)$ to denote its length and $m(r)$ its mode.

2. *A* trajectory *of $\mathcal{H}$ is a finite sequence of flows $\sigma = r_0, r_1, \dots, r_p$ such that:*
   - *$len(r_i) = l_i$ for all $i = 0, \dots, p$,*
   - *if $i > 0$, $(r_{i-1}(l_{i-1}), r_i(0)) \in Jump$ for all $i = 1, \dots, p$,*
   - *if $l_i > 0$ then for all $t \in [0, l_i]$, $(m(r), r(t), \dot{r}(t)) \in Flow$, where $\dot{r}$ is the derivative of the projection of r to its continuous part.*
3. *A hybrid system $\mathcal{H} = (Flow, Jump, Init, UnSafe)$ is safe if and only if, there is no trajectory $r_0, \dots, r_p$ of $\mathcal{H}$ such that $r_0(0)$ is in Init and $r_p(l)$ is in UnSafe, where l is the length of r_p.*

The semantics of a hybrid system is a transition system with an uncountable set of states. Formally, the semantics of a hybrid system $\mathcal{H} = (Flow, Jump, Init, UnSafe)$ is a transition system $\mathcal{M}(\mathcal{H}) = (S, S_{Init}, Steps, S_{UnSafe})$ where $S = \mathtt{M} \times I_1 \times \dots \times I_k$, $S_{Init} = \{s \in S : s \text{ satisfies } Init\}$, $S_{UnSafe} = \{s \in S : s \text{ satisfies } UnSafe\}$, and $Steps$ is defined as the union of two transition relations $Steps_{\mathcal{C}}$ and $Steps_{\mathcal{D}}$, where $Steps_{\mathcal{C}} \subseteq S \times S$ corresponds to transitions due to continuous flows and is defined by:

- $((m, \vec{x}), (m, \vec{x'})) \in Steps_{\mathcal{C}}$, if there exists a trajectory $\sigma = r_0$ (i.e., a flow r_0) such that $m(r_0) = m$, $r_0(0) = \vec{x}$ and $r(len(r_0)) = \vec{x'}$,

and $Steps_{\mathcal{D}} \subseteq S \times S$ corresponds to transitions due to discrete jumps and is defined by:

- $((m, \vec{x}), (m', \vec{x'})) \in Steps_{\mathcal{D}}$ if $((m, \vec{x}), (m', \vec{x'})) \in Jump$.

It is well-known that checking whether a hybrid system is safe is an undecidable problem (Henzinger et al., 1998). However, in practice we are not interested with a hybrid system whose safety changes under small perturbations. Hence, it is sufficient to have an algorithm that can prove safety for systems whose safety does not change under small perturbations.

In order to introduce the notation of perturbations, we first define a distance measure on constraints as follows.

Definition 4.

1. *The distance between two atomic arithmetic constraints $e\,r\,c$ and $e'\,r'\,c'$ is defined by $d(e, r, c, e', r', c') \doteq \infty$, if $e \neq e'$ or $r \neq r'$, and $|c - c'|$, otherwise.*
2. *The distance between two mode constraints $m = m_1$ and $m = m_2$ is ∞ if $m_1 \neq m_2$ and 0, otherwise.*
3. *The distance between two constraints ϕ and ϕ' is defined by $d(\phi, \phi') \doteq$*
   - *∞, if ϕ and ϕ' have a different Boolean structure or do not have mode constraints at the same place, and*
   - *the maximum of the distances between two corresponding atomic (arithmetic or mode) constraint, otherwise.*

Now, after denoting the distance between two vectors $\vec{x}$ and $\vec{x'}$ to be

$$d(\vec{x}, \vec{x'}) = \|\vec{x} - \vec{x'}\|_\infty = \max_{1 \le i \le k} |x_i - x'_i|.$$

based on Definition 4, we can define the notion of an ε-perturbed solution set as follows.

Definition 5. *A set P is an ε-perturbed solution set of a constraint ϕ if and only if*

1. *for every $\vec{x} \in P$, there is a constraint ϕ^* with $d(\phi, \phi^*) \le \varepsilon$ and an $\vec{x}^*$ with $d(\vec{x}, \vec{x}^*) \le \varepsilon$ such that $\phi^*(\vec{x}^*)$ holds.*
2. *for every $\vec{x} \notin P$, there is a constraint ϕ^* with $d(\phi, \phi^*) \le \varepsilon$ and an $\vec{x}^*$ with $d(\vec{x}, \vec{x}^*) \le \varepsilon$ such that $\phi^*(\vec{x}^*)$ does not hold.*

Example 1. *Consider the constraint ϕ defined by $x = 0$. Clearly, $\{x : x = 0\}$ is an ε-perturbed solution set due to the following:*

1. *for $x = 0$, choosing ϕ as ϕ^* and x as x^*, $d(\phi, \phi^*) = 0 < \varepsilon$, $d(x, x^*) = 0 < \varepsilon$, and $\phi^*(x^*)$ holds.*
2. *for any $x \neq 0$, choosing ϕ as ϕ^* and x as x^*, $d(\phi, \phi^*) = 0 < \varepsilon$, $d(x, x^*) = 0 < \varepsilon$, and $\phi^*(x^*)$ does not hold.*

Moreover, $P = \{x : x = \varepsilon\}$ with $\varepsilon > 0$ is also an ε-perturbed solution set due to:

1. *for $x = \varepsilon$, choosing $x = \varepsilon/2$ as ϕ^* and $x^* = \varepsilon/2$, $d(\phi, \phi^*) = \varepsilon/2 < \varepsilon$, $d(x, x^*) = \varepsilon/2 < \varepsilon$, and $\phi^*(x^*)$ holds.*
2. *for any $x \neq \varepsilon$, choosing $x = \varepsilon/2$ as ϕ^* and $x^* = x - \varepsilon/2$, $d(\phi, \phi^*) = \varepsilon/2 < \varepsilon$, $d(x, x^*) = \varepsilon/2 < \varepsilon$, and $\phi^*(x^*)$ does not hold.*

$\square$

Example 2. *Consider another constraint ϕ defined by $x^2 < 0$. Clearly, its solution set is empty and $\emptyset$ is an ε-perturbed solution set of ϕ. Moveover, $P = \{x : x^2 < \varepsilon\}$ is also an ε-perturbed solution set of ϕ due to:*

1. *for every $x \in P$, choosing $x^2 < \varepsilon$ as ϕ^* and $x^* = \min\{\varepsilon + x, \sqrt{\varepsilon}\}$, $d(\phi, \phi^*) = \varepsilon$, $d(x, x^*) \le \varepsilon$, and $\phi^*(x^*)$ holds.*
2. *for every x such that $x \ge \sqrt{\varepsilon}$, choosing $x^2 < \varepsilon$ as ϕ^* and $x^* = x + \varepsilon$, $d(\phi, \phi^*) = \varepsilon$, $d(x, x^*) = \varepsilon$, and $\phi^*(x^*)$ does not hold.*
3. *for every x such that $x \le -\sqrt{\varepsilon}$, choosing $x^2 < \varepsilon$ as ϕ^* and $x^* = x - \varepsilon$, $d(\phi, \phi^*) = \varepsilon$, $d(x, x^*) = \varepsilon$, and $\phi^*(x^*)$ does not hold.* $\square$

Definition 5 is extended for hybrid systems with small perturbations as follows.

Definition 6. *A hybrid system $\mathcal{H}_\varepsilon = (Flow_\varepsilon, Jump_\varepsilon, Init_\varepsilon, UnSafe_\varepsilon)$ is an ε-perturbed manifestation of a hybrid system $\mathcal{H} = (Flow, Jump, Init, UnSafe)$ if and only if $Flow_\varepsilon$, $Jump_\varepsilon$, $Init_\varepsilon$ and $UnSafe_\varepsilon$ are ε-perturbed solution sets of Flow, Jump, Init and UnSafe, respectively.*

Definition 6 allows us to define robustness of a hybrid system with the same desired safety property as follows.

Definition 7. *A hybrid system $\mathcal{H} = (Flow, Jump, Init, UnSafe)$ is robustly safe if and only if there exists a constant $\varepsilon > 0$ such that all its ε-perturbed manifestations $\mathcal{H}_\varepsilon = (Flow_\varepsilon, Jump_\varepsilon, Init_\varepsilon, UnSafe_\varepsilon)$ are safe.*

Example 3. *Consider the hybrid system $\mathcal{H} = (Flow, Jump, Init, UnSafe)$, where Flow is $\dot{x} = 0$, Jump is $\emptyset$, Init is $x = 0$, and UnSafe is $x = 1$. Clearly, this hybrid system $\mathcal{H}$ is safe. However, $\mathcal{H}' = (\dot{x} = \varepsilon, \emptyset, x = 0, x = 1)$ with $\varepsilon > 0$ is an ε-perturbed manifestation of $\mathcal{H}$ but not safe, implying that H is not robustly safe.* □

3 CONSTRAINT BASED ABSTRACTION REFINEMENT

In this section we describe a constraint based algorithm for safety verification of hybrid systems based on the abstraction refinement technology. That is, we abstract a hybrid system to a finite transition system (the *abstraction*) which is defined to be:

Definition 8. *A* transition system *over a finite set Σ is a tuple (Trans, Init, Unsafe) where $Trans \subseteq \Sigma \times \Sigma$ and $Init \subseteq \Sigma$, $Unsafe \subseteq \Sigma$. We call the set Σ the* state space *of the system.*

In contrast to Definition 2, here the state space is a parameter. This will allow us to add/remove states to the state space during abstraction refinement.

Definition 9. *A* trajectory *of a* transition system $(Trans, Init, UnSafe)$ over a set Σ *is a function $r : \{0, \dots, p\} \mapsto \sigma$ such that for all $t \in \{1, \dots, p\}$, $(r(t-1), r(t)) \in Trans$. The system is* safe *if and only if there is no trajectory from an element of Init, to an element of Unsafe.*

When we use abstraction to analyze hybrid systems, the abstraction should over-approximate the concrete system in a conservative way: if the abstraction is safe, then the original system should also be safe. If the current abstraction is not yet safe, we refine the abstraction, that is, we include more information about the concrete system into it. This results in Algorithm 1.

Algorithm 1: Abstraction Refinement.

```
Input:   a hybrid system $\mathcal{H}$ described by constraints
Output:  "safe", if the algorithm terminates
  let $A$ be an abstraction of the hybrid system $\mathcal{H}$
  while $A$ is not safe do
    refine the abstraction $A$
  end while
```

In order to implement this algorithm, we need to fix the state space of the abstract system. Here we use pairs (m, B), where m is one of the modes $\{m_1, \dots, m_n\}$ and B is a hyper-rectangle (*box*), representing subsets of the concrete state space S. Together with an abstract state, we store the information whether it is initial or unsafe and the information from which other states it is reachable. We call such information the *marks* of the state. For the initial abstraction we use the state space $\{(m_i, \{\vec{x} \mid (m_i, \vec{x}) \in S\}) \mid 1 \le i \le n\}$, where all states are marked as initial, and unsafe, and all transitions between states are possible.

For refining the abstraction, we split a box into two pieces, replace one abstract state by two, and include more information from the concrete system into the abstract one by removing unreachable elements from the boxes, removing superfluous marks from the new abstract states, and removing unreachable states from the abstraction.

To remove unreachable elements from the boxes representing the abstraction, we use a constraint that formalizes when an element of the concrete state space might be reachable, and then remove elements that do not fulfill this constraint. In order to do this, for a box $B = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_k, \overline{x}_k]$, we let its j-th lower face be $[\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_j, \underline{x}_j] \times \cdots \times [\underline{x}_k, \overline{x}_k]$ and its j-th upper face be $[\underline{x}_1, \overline{x}_1] \times \cdots \times [\overline{x}_j, \overline{x}_j] \times \cdots \times [\underline{x}_k, \overline{x}_k]$. Note that two boxes in the same mode are *non-overlapping* if their interiors are disjoint.

Observe that a point in a box B is reachable only if it is reachable either from the initial set via a flow in B, from a jump via a flow in B, or from a neighboring box via a flow in B. So we can formulate constraints corresponding to each of these conditions and then remove points from boxes that do not fulfill at least one of these constraints. For this, we first give a constraint describing flows within boxes as follows, which has been described in (Ratschan and She, 2006).

Lemma 1. *For a box $B \subseteq \mathbb{R}^k$ and a mode m, if there is a flow in B and m from a point $\vec{x} = (x_1, \dots, x_k)^T \in B$ to* *a point $\vec{y} = (y_1,\ldots,y_k)^T \in B$ such that for every point $\vec{u}$ on the flow with its derivative $\dot{\vec{u}}$, $(m,\vec{u},\dot{\vec{u}})$ satisfies the flow constraint $Flow(m,\vec{x},\dot{\vec{x}})$, then*

$$\exists t \in \mathbb{R}_{\geq 0}[flow^*_B(t,\vec{x},\vec{y})], \tag{1}$$

where $flow^*_B(t,\vec{x},\vec{y})$ denotes

$$\bigwedge_{1\leq i\leq k} \exists a_1,\ldots,a_k,\dot{a}_1,\ldots,\dot{a}_k[(a_1,\ldots,a_k) \in B$$
$$\wedge Flow(m,(a_1,\ldots,a_k),(\dot{a}_1,\ldots,\dot{a}_k)) \wedge y_i = x_i + \dot{a}_i \cdot t]$$

We denote the above constraint by $Reach_B(m,\vec{x},\vec{y})$. Notice that in Lemma 1, the state $(m,\vec{y})$ is assumed to be reachable from $(m,\vec{x})$ via a flow in B. However, the information on the state $(m,\vec{x})$ is missing, which in fact requires to be reachable via a trajectory starting from initial. Without loss of generality, we can beforehand assume that we already have a constraint $Reachable_B(m,\vec{x})$ describing that $(m,\vec{x})$ is reachable from initial. Thus, the above three possibilities for reachability allow us to formulate the following theorem:

Theorem 1. *For a set of abstract states $\mathcal{B}$, a pair $(m',B') \in \mathcal{B}$ and a point $\vec{z} \in B'$, if $(m',\vec{z})$ is reachable and z is not an element of the box of any other abstract state in $\mathcal{B}$, then*

$$Ifl_{B'}(m',\vec{z}) \vee \bigvee_{(m,B)\in\mathcal{B}} Jfl_{B,B'}(m,m',\vec{z})$$
$$\vee \bigvee_{(m,B)\in\mathcal{B},m=m',B\neq B'} Bfl_{B,B'}(m',\vec{z})$$

where $Ifl_{B'}(m',\vec{z})$, $Jfl_{B,B'}(m,m',\vec{z})$, and $Bfl_{B,B'}(m',\vec{z})$ denote the following three constraints, respectively:

- $\exists\vec{x} \in B'[Init(m',\vec{x}) \wedge Reach_{B'}(m',\vec{x},\vec{z})]$,
- $\exists\vec{x} \in B\exists\vec{x}' \in B'[Reachable_B(m,\vec{x}) \wedge Jump(m,\vec{x},m',\vec{x}') \wedge Reach_{B'}(m',\vec{x}',\vec{z})]$
- $\exists\vec{x} \in B\cap B'[Reachable_B(m,\vec{x}) \wedge [\forall faces\, F\, of\, B'[\vec{x} \in F \Rightarrow in^F_{m',B'}(\vec{x})]] \wedge Reach_{B'}(m',\vec{x},\vec{z})]$.

Here, $in^F_{m',B'}(\vec{x}) = \exists\dot{x}_1,\ldots,\exists\dot{x}_k[Flow(m',\vec{x},(\dot{x}_1,\ldots,\dot{x}_k)) \wedge \dot{x}_j \geq 0]$ if F is the j-th lower face of B', and $in^F_{m',B'}(\vec{x}) = \exists\dot{x}_1,\ldots,\exists\dot{x}_k[Flow(m',\vec{x},(\dot{x}_1,\ldots,\dot{x}_k)) \wedge \dot{x}_j \leq 0]$ if F is the j-th upper face of B'.

Based on Theorem 1, if we can prove that a certain point does not fulfill the big constraint in Theorem 1, we know that it is not reachable from the set of initial states. However, the big constraint is not first-order, since it uses some defined predicates (e.g., $Reachable_B$ and faces F of B'). Thus, we need to eliminate the defined predicates by substituting the constraints implied by their definitions. For this, we first have to fix a certain constraint for $Reachable_B(m,\vec{x})$.

As an over-approximation, the first and simplest choice is to define the constraint $Reachable_B(m,\vec{x})$ as $\vec{x} \in B$ and denote the resulting constraint by $Reachable^0_{\mathcal{B},B'}(m',\vec{z})$. We have studied this case with computational examples in (Ratschan and She, 2005; Ratschan and She, 2007; Ratschan and She, 2007a; Ratschan and She, 2006) by using a *pruning algorithm* (Ratschan, 2002) that takes a constraint, and an abstract state (m',B') and returns a sub-box of B' that still contains all the solutions of the constraint in B'.

By using the *pruning algorithm*, we can get a new B' by removing certain points that are not reachable and thus do not fulfill the constraint $Reachable^0_{\mathcal{B},B'}(m',\vec{z})$. Since the constraint $Reachable^0_{\mathcal{B},B'}(s',\vec{z})$ depends on all current abstract states, a change of B' might allow further pruning of other abstract states. So we can repeat pruning until a fixpoint is reached. For a given set of abstract states $\mathcal{B}$, we denote the resulting fixpoint by $Prune_{\mathcal{H}}(\mathcal{B})$. Moreover, since we do not need to consider unreachable parts of the state space in the abstraction, we can do the operation $\mathcal{B} \leftarrow Prune_{\mathcal{H}}(\mathcal{B})$ anywhere in Algorithm 1. We do this at the beginning, and each time $\mathcal{B}$ is refined by splitting a box.

Thus, the abstraction over $Prune_{\mathcal{H}}(\mathcal{B})$ for $\mathcal{H}$ is constructed as follows:

1. mark an abstract state (m',B') as initial if we cannot disprove $Ifl_{B'}(m',\vec{z})$ in Theorem 1;
2. mark an abstract state (m',B') as unsafe if we cannot disprove the constraint $\exists\vec{x} \in B' UnSafe(m',\vec{x})$;
3. let $Trans = Trans_{\mathcal{C}} \cup Trans_{\mathcal{D}}$, where $Trans_{\mathcal{C}}$ corresponds to the transitions due to the continuous flows, $Trans_{\mathcal{D}}$ corresponds to the the transitions due to the discrete jumps, and $Trans_{\mathcal{C}}$ and $Trans_{\mathcal{D}}$ are computed according to the following cases:
   - $m = m'$ and $B = B'$: $((m,B),(m,B)) \in Trans_{\mathcal{C}}$;
   - $m = m'$, $B \neq B'$ and we cannot disprove the constraint $Bfl_{m,B,B'}$ in Theorem 1: $((m,B),(m,B')) \in Trans_{\mathcal{C}}$;
   - If we cannot disprove the constraint $\exists x \in B\exists x' \in B' Jfl_{B,B'}(m,x,m',x')$ in Theorem 1: $((m,B),(m',B')) \in Trans_{\mathcal{D}}$.

The above abstraction is easily computed since the set of abstract states is finite. Denoting the resulting transition system by $Abstract_{\mathcal{H}}(\mathcal{B})$, we have that:

Theorem 2. *(Ratschan and She, 2007) For a hybrid system $\mathcal{H}$ and sets of abstract states $\mathcal{B}$, containing all elements of the state space reachable from initial, such that all boxes corresponding to the same mode are non-overlapping, the safety of $Abstract_{\mathcal{H}}(\mathcal{B})$ implies the safety of the hybrid system $\mathcal{H}$.*

Based on the first choice, our second choice is to substitute $Reachable^0_{\mathcal{B},B}(m,\vec{x})$ for $Reachable_B(m,\vec{x})$ in the big constraint in Theorem 1, resulting in a constraint $Reachable^1_{\mathcal{B},B'}(m',\vec{z})$.

Similarly, due to recursion, we have constraints $Reachable^i_{\mathcal{B},B'}(m',\vec{z})$ with $i \in \{0,\dots\}$. Clearly, all these constraints also fulfill Theorem 1.

Since $Reachable^i_{\mathcal{B},B'}(m',\vec{z})$ with $i \geq 1$ is a very large constraint, we will avoid directly disproving such a constraint by computing an over-approximation of the reach set on the boundary of B, arriving at the following constraint that expresses a disjunction over all faces:

$$\bigvee_{F,\text{face of } B} \left[\vec{x} \in F \wedge Reachable^0_{\mathcal{B},B}(m',\vec{x})\right].$$

We denote it by $reachbound_{\mathcal{B},B}(m',\vec{x})$, and for each face F of B, we denote the corresponding disjunct by $reachbound_{\mathcal{B},B,F}(m',\vec{x})$. Since the disjuncts only depend on one box, we have to apply the pruning algorithm only for one abstract state, and we can store the resulting faces with that abstract state and use such information in the constraint $Bfl_{B,B'}(m',\vec{z})$. Since these faces enclose the set of states where a flow might leave the abstract state, we call them the *outflow-faces* of the abstract state (cf., the use of faces in the analysis of rectangular automata (Preußig et al., 1998) and the use of faces for CEGAR based reachable analysis in Algorithm 4 in (Klaedtke et al., 2007)). Note that this recursive reasoning based method with computational examples has also been described in (Ratschan and She, 2008) and implemented in (Ratschan and She, 2007a).

3.1 A Remedy for Reducing the Wrapping Effect

We have introduced a recursive reasoning based method for safety verification of hybrid systems above. However, box splitting in some cases will still lead to an worse over-approximation (Preußig et al., 1998). This phenomenon is called as the wrapping effect (Neumaier, 1993), which is illustrated by the following example.

Example 4. *Consider a box* $[0,2]\times[0,2]\times[0,4]$ *and let the initial set be* $F_I = \{0 \leq x \leq 1 \wedge y = 0 \wedge t = 0\}$ *and the flow constraint be* $\phi = \{\dot{x} = \dot{y} = \dot{t} = 1\}$.

Clearly, the exact reachable set on the face is $F_E = \{x = 2 \wedge y = t \wedge 1 \leq y \leq 2\}$.

However, after using the recursive version the original box, we get a box $[0,2]\times[0,2]\times[0,2]$ *as the over-approximative reachable set. The over-approximative reachable set on the face is* $F_{R,1} = \{x = 2 \wedge 1 \leq y \leq 2 \wedge 1 \leq t \leq 2\}$.

Spilt $[0,2]\times[0,2]\times[0,2]$ *into the following four boxes:*

$$B_1 = [0,1]\times[0,1]\times[0,2], B_2 = [0,1]\times[1,2]\times[0,2],$$
$$B_3 = [1,2]\times[0,1]\times[0,2], B_4 = [1,2]\times[1,2]\times[0,2].$$

Using our recursive version again, we get the following three boxes:

$$B'_1 = [0,1]\times[0,1]\times[0,1],$$
$$B'_3 = [1,2]\times[0,1]\times[0,2],$$
$$B'_4 = [1,2]\times[1,2]\times[0,2].$$

Associated with these three boxes, we have the following faces:

$$F_I = \{0 \leq x \leq 1 \wedge y = 0 \wedge t = 0\};$$
$$F_1 = \{x = 1 \wedge 0 \leq y \leq 1 \wedge 0 \leq t \leq 1);$$
$$F_2 = \{1 \leq x \leq 2 \wedge y = 1 \wedge 0 \leq t \leq 2);$$
$$F_{R,2} = \{x = 2 \wedge 1 \leq y \leq 2 \wedge 0 \leq t \leq 2).$$

Clearly, $F_E \subset F_{R,1} \subset F_{R,2}$. □

So, we will in this subsection provide a remedy for reducing the wrapping effect. Specifically, we improve the recursive reasoning by four complements which are described in details as follows:

1. For each abstract $(m,B) \in \mathcal{B}$, we first compute the intervals for the corresponding components of the derivatives of the states in (m,B). That is, we apply our pruning algorithm to the following constraint and $\mathbb{R}^k$:
$$\exists a_1,\dots,a_k\left[(a_1,\dots,a_k) \in B \wedge Flow(m,(a_1,\dots,a_k),(\dot{a}_1,\dots,\dot{a}_k))\right] \quad (2)$$
to obtain a box containing all the solutions on $\dot{a}_1,\dots,\dot{a}_k$ satisfying the above constraint. We denote this resulting box by $\dot{B}$. This information has in fact been computed when we apply our pruning algorithm to the big constraint in Theorem 1 and (m,B). So, we just need to store this computed information and do not need to apply our pruning algorithm once more.

2. For each (m,B), letting $\dot{B} = \dot{B}_1 \times \cdots \times \dot{B}_k = [\underline{\dot{a}_1},\overline{\dot{a}_1}] \times \cdots \times [\underline{\dot{a}_k},\overline{\dot{a}_k}]$ in $Reach_B(m,\vec{x},\vec{y})$, we arrive at
$$\exists t \in \mathbb{R}_{\geq 0}\Big[\bigwedge_{1\leq i\leq k} \exists \dot{a}_i[\underline{\dot{a}_i} \leq \dot{a}_i \leq \overline{\dot{a}_i} \wedge y_i = x_i + \dot{a}_i \cdot t]\Big]. \quad (3)$$
We denote this constraint by $Reach'_B(m,\vec{x},\vec{y})$. Then, we use $Reach'_B(m,\vec{x},\vec{y})$ instead of $Reach_B(m,\vec{x},\vec{y})$ in $reachbound_{\mathcal{B},B,F}(m',\vec{x})$ to get a new constraint, which is denoted by $reachbound'_{\mathcal{B},B,F}(m',\vec{x})$.

3. For every predicate of form $\vec{x} \in B$ in $\mathit{reachbound}'_{\mathcal{B},B,F}(m',\vec{x})$, where $B = [\underline{x_1},\overline{x_1}] \times \cdots \times [\underline{x_k},\overline{x_k}]$, we will replace it by $\bigwedge_{i=1}^{k} \underline{x_i} \le x_i \le \overline{x_i}$, arriving at the constraint $\mathit{reachbound}''_{\mathcal{B},B,F}(m',\vec{x})$. Then, we use a quantifier elimination method to $\mathit{reachbound}''_{\mathcal{B},B,F}(m',\vec{x})$ to compute the reachable states on the face F, which will be discussed in Subsection 3.2. Note that in this way, for every face F, we get the exact solutions of $\vec{x}$ to the constraint $\mathit{reachbound}'_{\mathcal{B},B,F}(m',\vec{x})$, which describes the reachable information on the face F. Moreover, note that for abstract states marked as initial or reachable with jumps, we do not need to recompute the exact solution set on the faces but directly use the over-approximations instead, which can be easily understood after termination analysis in Section 4.
4. We use the reachable information on the faces in the constraint $\mathit{Bfl}_{B,B'}(m',\vec{z})$ for further computation until a fixpoint is reached.

Now, for Example 4, starting with the initial abstraction, after applying the above improvement, the over-approximative reachable set on the face is computed to be

$$F_R = \{x = 2 \wedge y = t \wedge 1 \le y \le 2\},$$

which is the exact reachable set F_E.

Clearly, except for the initial constraint and the jump constraint, every arithmetic expression in the constraint $\mathit{reachbound}''_{\mathcal{B},B,F}(m',\vec{x})$ is either a linear equality or a linear inequality. Hence, the solution set to $\mathit{reachbound}''_{\mathcal{B},B,F}(m',\vec{x})$ can be easily computed by the quantifier elimination. Note that we do not directly apply the quantifier elimination to $\mathit{reachbound}_{\mathcal{B},B,F}(m',\vec{x})$ due to the fact that the flow constraint used in $\mathit{Reach}'_B(m,\vec{x},\vec{y})$ may be non-polynomial.

Thus, based on the above four complements, letting $\mathit{Abstract}'_{\mathcal{H}}(\mathcal{B})$ be the resulting system, we have the following theorem which is similar to Theorem 2.

Theorem 3. *For a hybrid system $\mathcal{H}$ and sets of abstract states $\mathcal{B}$, containing all elements of the state space reachable from the initial set such that all boxes corresponding to the same mode are non-overlapping, the safety of $\mathit{Abstract}'_{\mathcal{H}}(\mathcal{B})$ implies the safety of the hybrid system $\mathcal{H}$.*

3.2 Quantifier Elimination

In this subsection, we will discuss how to apply the quantifier elimination mentioned in Subsection 3.1.

We first assume that $\dot{B} = \dot{B}_1 \times \cdots \times \dot{B}_k$ is the box obtained by applying the pruning algorithm to Constraint (2) and $\mathbb{R}^k$. Let I be the set $\{i : 0 \in \dot{B}_i\}$. Without loss of generality, let $I = \{i_1,\ldots,i_m\}$ and consider the following constraint:

$$\exists t \in \mathbb{R}_{\ge 0}\Big[\Big[\bigwedge_{\substack{1 \le j \le k \\ j \notin \{i_1,\ldots,i_m\}}} \exists \dot{a}_j[\underline{\dot{a}_j} \le \dot{a}_j \le \overline{\dot{a}_j} \wedge y_j = x_j + \dot{a}_j \cdot t]\Big] \bigwedge \Big[\bigwedge_{j \in \{i_1,\ldots,i_m\}} \exists \dot{a}_j[\underline{\dot{a}_j} \le \dot{a}_j \le \overline{\dot{a}_j} \wedge y_j = x_j + \dot{a}_j \cdot t]\Big]\Big]. \tag{4}$$

Clearly, t in Constraint (4) can be easily eliminated. Moreover, x_i and y_i satisfying the constraint are in a polyhedron defined by a combination of linear (in)equalities over $\vec{x}$ and $\vec{y}$.

Specifically, this combination has

1. $(k-m)(k-m-1)/2$ (in)equalities of form
   - $\underline{\dot{a}_i}/\overline{\dot{a}_j}(y_j - x_j) \le (y_i - x_i) \le \overline{\dot{a}_i}/\underline{\dot{a}_j}(y_j - x_j)$, if $\dot{a}_i\dot{a}_j > 0$, or
   - $\underline{\dot{a}_i}/\underline{\dot{a}_j}(y_j - x_j) \le (y_i - x_i) \le \overline{\dot{a}_i}/\overline{\dot{a}_j}(y_j - x_j)$, if $\dot{a}_i\dot{a}_j < 0$,

   where $i, j \in \{1,\ldots,k\} \setminus \{i_1,\ldots,i_m\}, i \ne j$,
2. $(k-m)m$ (in)equalities of form
   - $\underline{\dot{a}_i}/\underline{\dot{a}_j}(y_j - x_j) \le (y_i - x_i) \le \overline{\dot{a}_i}/\underline{\dot{a}_j}(y_j - x_j)$, if $\dot{a}_j > 0$, or
   - $\underline{\dot{a}_i}/\overline{\dot{a}_j}(y_j - x_j) \le (y_i - x_i) \le \overline{\dot{a}_i}/\overline{\dot{a}_j}(y_j - x_j)$, if $\dot{a}_j < 0$,

   where $j \in \{1,\ldots,k\} \setminus \{i_1,\ldots,i_m\}, i \in \{i_1,\ldots,i_m\}$, and
3. $(k-m)$ (in)equalities of form $y_i - x_i \le 0$ or $y_i - x_i \ge 0$, which are determined by the signs of the rate intervals.

If the solution set for $\vec{x}$ is defined by a polyhedron which is formulated by a combination of linear (in)equalities over $\vec{x}$, the solutions set for $\vec{y}$ will also be formulated by a combination of linear (in)equalities over $\vec{y}$, implying that the solution set for $\vec{y}$ is also defined by a polyhedron.

4 TERMINATION ANALYSIS FOR SAFETY VERIFICATION

In this section, we will analyze the termination of our abstraction refinement based procedure for robustly safe hybrid systems, associated with the remedy described in Subsection 3.1. Note that we here simply assume that the continuous behaviors evolve according to differential equations, that is, we only consider deterministic continuous evolutions. For the non-deterministic cases, we can similarly handle it.

Without loss of generality, we assume that in the flow constraint, the right side of the implication is of form $\dot{\vec{x}} - f_m(\vec{x}) = 0$, where f_m is Lipschitz on $I_1 \times \cdots \times I_k$ with the Lipschitz constant L_m. Let $L = \max_{m \in M} L_m$. In addition, let $d(B) = \max_{\vec{x},\vec{x}' \in B} d(\vec{x},\vec{x}')$ and $d(\mathcal{B}) = \max_{B \in \mathcal{B}} d(B)$. Moreover, for two sets A and B with $B \subseteq A$, let $d(A,B) = \sup_{\vec{x} \in A} \inf_{\vec{x}' \in B} d(\vec{x},\vec{x}')$.

For every mode m and box B, let $\phi(m,B)$ be the solution set of $\phi_{(m,B)}$ in $\mathbb{R}^k$ and $Prune(\phi_{m,B},\mathbb{R}^k)$ be the result of applying our interval based pruning algorithm to $\phi_{(m,B)}$ and $\mathbb{R}^k$. Due to the convergence of the interval based pruning algorithm (Ratschan, 2002; Damm et al., 2007), we have:

Theorem 4. *(Ratschan, 2002; Damm et al., 2007) For every mode m and box B,*

$$\lim_{d(B)\to 0} d(Prune(\phi_{(m,B)},\mathbb{R}^k),\phi(m,B)) = 0.$$

Assume that the system $\mathcal{H} = (Flow, Jump, Init, UnSafe)$ is robustly safe, that is, there is an ε such that all its ε-perturbed manifestations $\mathcal{H}_\varepsilon$ are also safe. In addition, we assume that for each refinement step, the abstraction computed by our improved procedure is $Abstract'_{\mathcal{B}}(\mathcal{H}) = (Trans', Init', UnSafe')$ with $Trans' = Trans'_C \cup Trans'_D$, where $Trans'_C$ corresponds to flows and $Trans'_D$ corresponds to jumps.

Due to Theorem 4, for the given constant $\varepsilon/3$, there is a σ_1 such that when $d(B) \le \sigma_1$, $d(Prune(\phi,(m,B)),\phi(m,B)) \le \varepsilon/3$. Letting $\varepsilon_1 = \min\{\sigma_1, \varepsilon/3\}$, we can beforehand assume that $d(\mathcal{B}) \le \sigma = \min\{\frac{\varepsilon_1}{kL}, \varepsilon_1\}$.

Without loss of generality, let us first assume that the solution set of $Jump(m,\vec{x},m',\vec{x}')$ is empty.

Let $Init_\varepsilon$ be the set $\{(m,\vec{x}) : d(\vec{x},\vec{x}') \le \varepsilon, Init(m,\vec{x}')\}$. Clearly, $Init \subset Init_\varepsilon$. Moreover, $Init_\varepsilon$ is an ε-perturbed solution set of $Init$ due to

1. for every $\vec{x} \in Init_\varepsilon$, choosing $Init$ as ϕ^* and an $\vec{x}'$ such that $d(\vec{x},\vec{x}') \le \varepsilon$ and $Init(m,\vec{x}')$ as $\vec{x}^*$, $d(Init,\phi^*) = 0 \le \varepsilon$ and $\phi^*(m,\vec{x}^*)$ holds.
2. for every $\vec{x} \notin Init_\varepsilon$, choosing $Init$ as ϕ^* and $\vec{x}$ as $\vec{x}^*$, $d(Init,\phi^*) = 0 \le \varepsilon$, $d(\vec{x},\vec{x}^*) = 0 \le \varepsilon$ and $Init(m,\vec{x})$ does not hold.

Similarly, let $UnSafe_\varepsilon$ be the set $\{(m,\vec{x}) : d(\vec{x},\vec{x}') \le \varepsilon, UnSafe(m,\vec{x}')\}$. Then, $UnSafe \subset UnSafe_\varepsilon$ and $UnSafe_\varepsilon$ is an ε-perturbed solution set of $UnSafe$.

Let $Flow_\varepsilon$ be the set $\{(m,\vec{x},\dot{\vec{x}}) : d((\vec{x},\dot{\vec{x}}),(\vec{x}',\dot{\vec{x}}')) \le \varepsilon, Flow(m,\vec{x}',\dot{\vec{x}}')\}$. Then, $Flow \subset Flow_\varepsilon$ and $Flow_\varepsilon$ is an ε-perturbed solution set of $Flow$.

Now, let $\mathcal{H}_\varepsilon = (Init_\varepsilon, Flow_\varepsilon, \emptyset, UnSafe_\varepsilon)$.[1] Clearly, $\mathcal{H}_\varepsilon$ is an ε-perturbed manifestation of $\mathcal{H}$. Similarly, we can define $\mathcal{H}_{\varepsilon_1}$ and $\mathcal{H}_{\varepsilon/3}$.

In addition, let $ReachSet_{\varepsilon_1}$, $ReachSet_{\varepsilon/3}$, and $ReachSet_\varepsilon$ be the reachable set of $\mathcal{H}_{\varepsilon_1}$, $\mathcal{H}_{\varepsilon/3}$, and $\mathcal{H}_\varepsilon$. Clearly, $ReachSet_{\varepsilon_1} \subseteq ReachSet_{\varepsilon/3} \subseteq ReachSet_\varepsilon$.

Since we are just interested with the abstract states that are reachable from $Init'$ instead of reachable by possible loops (or, cycles), it is sufficient to only consider the abstract states that can be reachable from $Init'$. That is, letting

$$RS = \{(m,\vec{x}) | (m,\vec{x}) \in (m,B), (m,B) \in \mathcal{B}, \text{ and } (m,B) \text{ can be reachable from } Init'\},$$

for terminating analysis, it is sufficient to prove that $RS \subseteq ReachSet_\varepsilon$.

For proving that $RS \subseteq ReachSet_\varepsilon$, we first introduce some notations as follows:

1. For each face F' of (m',B'), let $S_{B',F'}$ be the solution set of $reachout'_{\mathcal{B},B',F'}(m',\vec{z})$ in F' by applying the quantifier elimination method introduced in Subsection 3.2.
2. For an arbitrary but fix state $(m,\vec{x})$, let $B((m,\vec{x}),\sigma)$ be the set
$$\{(m,\vec{z}) : d((m,\vec{z}),(m,\vec{x})) \le \sigma\}.$$
3. For an arbitrary but fix set (m,S) of states , let $B((m,S),\sigma)$ be the set
$$\{(m,\vec{z}) : \exists \vec{x} \in S[d((m,\vec{z}),(m,\vec{x})) \le \sigma]\}.$$

Due to the definitions of $Init_\varepsilon$ and $Flow_\varepsilon$, we have the following two lemmas.

Lemma 2. *For each $(m,B) \in Init'$, $(m,B) \subseteq Init_{\varepsilon_1} \subseteq Init_\varepsilon$.*

Proof. Clearly, $Init_{\varepsilon_1} \subseteq Init_\varepsilon$. So, for each $(m,B) \in Init'$, we only need to prove that for each $(m,B) \in Init'$, $(m,B) \subseteq Init_{\varepsilon_1}$. For this, it is sufficient to prove that for each $(m,\vec{x}) \in (m,B)$, $(m,\vec{x}) \in Init_{\varepsilon_1}$.

Since $(m,B) \in Init'$, there is a $\vec{x}^* \in B$ such that $Init(m,\vec{x}^*)$ holds. Moreover, since $\vec{x},\vec{x}^* \in B$ and $d(B) \le \varepsilon_1$, we have $d(\vec{x},\vec{x}^*) \le \varepsilon_1$. Due to the definition of $Init_{\varepsilon_1}$, we proved that for each $(m,\vec{x}) \in (m,B)$, $(m,\vec{x}) \in Init_{\varepsilon_1}$. □

Lemma 3. *For each pair $(m',B') \in \mathcal{B}$, assume that (m',B') is reachable from $Init'$ via finite transitions in $Trans'_C$. Then, for each face F' of (m',B'), $S_{B',F'} \subseteq ReachSet_\varepsilon$ if $S_{B',F'} \neq \emptyset$. Moreover, $B(S_{B',F'},\sigma) \subseteq ReachSet_\varepsilon$, which implies that $(m',B') \subseteq ReachSet_\varepsilon$.*

[1]In $\mathcal{H}_\varepsilon$, due to the definition of $Flow_\varepsilon$, the continuous evolutions are non-deterministic.

Proof. We will prove it by the induction method.

- Letting $(m,B) \in Init'$, we want to prove that for each face F of (m,B), $F = S_{B,F} \subseteq ReachSet_\varepsilon$ and $B(F,\sigma) \subseteq ReachSet_\varepsilon$. According to Lemma 2, $(m,B) \subseteq Init_{\varepsilon_1} \subseteq ReachSet_\varepsilon$, which implies that for each face F of (m,B), $S_{B,F} \subseteq ReachSet_\varepsilon$ and $B(F,\sigma) \subseteq ReachSet_\varepsilon$.
- Assume that (m,B) is reachable from $Init'$ such that for each face F of (m,B), $(m,S_{B,F}) \subseteq ReachSet_\varepsilon$ and $B(F,\sigma) \subseteq ReachSet_\varepsilon$ when $S_{B,F} \neq \emptyset$. First, we want to prove that if $((m,B),(m',B')) \in Trans'_C$, then for each face F' of (m',B'), $S_{B',F'} \subseteq ReachSet_\varepsilon$ when $S_{B',F'} \neq \emptyset$. According to Theorem 1 and Eq. (3), if $S_{B',F'}$ is not empty, then for each $(m,\vec{y}) \in S_{B',F'}$, there is a state $(m,\vec{x}) \in S_{B,F}$ such that $Reach_{B'}(m,\vec{x},\vec{y})$ holds. That is, the following constraint holds:

$$\exists t \in \mathbb{R}_{\geq 0}\Big[\bigwedge_{1\leq i\leq k} \exists \dot{a}_i[\underline{\dot{a}_i} \leq \dot{a}_i \leq \overline{\dot{a}_i} \wedge y_i = x_i + \dot{a}_i \cdot t]\Big]. \tag{5}$$

So, it is sufficient to prove that $(m,\vec{y}) \in ReachSet_\varepsilon$. Equivalently, letting $\dot{x}_i = \dot{a}_i$, it is sufficient to prove that $d((\dot{a}_1,\ldots,\dot{a}_k), f(\vec{y})) \leq \varepsilon$, which can be completed by combining the following arguments:

  - Let $\underline{A_i} = \inf\{\dot{a}_i : \dot{a}_i = f_{m,i}(\vec{a}), \vec{a} \in B'\}$ and $\overline{A_i} = \sup\{\dot{a}_i : \dot{a}_i = f_{m,i}(\vec{a}), \vec{a} \in B'\}$. Moreover, let $\vec{y} \in B'$ be such that $\overline{A_i} = f_{m,i}(\vec{y})$ and $\vec{x} \in B'$ be such that $\underline{A_i} = f_{m,i}(\vec{x})$. Since f_m is Lipschitz with the Lipschitz constant L_m, $d(\overline{A_i},\underline{A_i}) = \| f_{m,i}(\vec{y}) - f_{m,i}(\vec{x})\| \leq L\|y-x\|_2 \leq kLd(y,x) \leq \varepsilon_1$, when $d(B') \leq \sigma = \min\{\frac{\varepsilon_1}{kL},\varepsilon_1\}$.
  - Let $\dot{B} = [\underline{\dot{a}_1},\overline{\dot{a}_1}] \times \cdots \times [\underline{\dot{a}_k},\overline{\dot{a}_k}]$ be the result of applying our interval based pruning algorithm. Since $d(B') \leq \sigma$, due to Theorem 4 and our assumptions, $d(\underline{\dot{a}_i},\underline{\dot{A}_i}) \leq \varepsilon/3$ and $d(\overline{\dot{a}_i},\overline{\dot{A}_i}) \leq \varepsilon/3$.
  - If $\dot{a}_i \in [\underline{a_i},\underline{A_i}]$, then $d(\dot{a}_i, f_{m,i}(\vec{y})) \leq d(\dot{a}_i,\underline{A_i}) + d(\underline{A_i}, f_{m,i}(\vec{y})) \leq \varepsilon/3 + \varepsilon_1$; If $\dot{a}_i \in [\overline{A_i},\overline{a_i}]$, then $d(\dot{a}_i, f_{m,i}(\vec{y})) \leq d(\dot{a}_i,\overline{A_i}) + d(\overline{A_i}, f_{m,i}(\vec{y})) \leq \varepsilon/3 + \varepsilon_1$; And if $\dot{a}_i \in [\underline{A_i},\overline{A_i}]$, then $d(\dot{a}_i, f_{m,i}(\vec{y})) \leq d(\underline{A_i},\overline{A_i}) \leq \varepsilon_1$.

  Thus, due to the above arguments, we proved that for each face $F'$of (m',B'), $S_{B',F'} \subseteq ReachSet_{3\varepsilon_1}$ when $S_{B',F'} \neq \emptyset$.
  Second, we want to prove that $B(S_{B',F'},\sigma) \subseteq ReachSet_\varepsilon$. Since for every arbitrary but fix state $(m',\vec{y}') \in B(S_{B',F'},\sigma)$, there exists a state $(m',\vec{y}) \in S_{B',F'}$ such that $d(\vec{y},\vec{y}') \leq \sigma$. According to the definition of $S_{B',F'}$, there is a state $(m,\vec{x}) \in S_{B,F}$ such that $(m',\vec{y})$ is reachable from $(m,\vec{x})$ via a flow determined by $\dot{\vec{x}} = \dot{\vec{a}}$, where $\dot{\vec{a}}_i$, $i = 1,\ldots k$, is the same as the one occurs in Eq.(5). Clearly, $(m',\vec{y}')$ can be reachable from a state $(m,\vec{x}+\vec{y}'-\vec{y})$ via the flow determined by $\dot{\vec{x}} = \dot{\vec{a}}$. Since $B(S_{B,F},\sigma) \subseteq ReachSet_\varepsilon$,

$$d(f_{m'}(\vec{y}), f_{m'}(\vec{y}')) \leq kLd(\vec{y},\vec{y}') \leq \varepsilon_1$$

  when $d(\mathcal{B}) \leq \sigma = \min\{\frac{\varepsilon_1}{kL},\varepsilon_1\}$, and thus $d(\dot{\vec{a}}, f_{m'}(\vec{y}')) \leq d(\dot{\vec{a}}, f_{m'}(\vec{y})) + d(f_{m'}(\vec{y}), f_{m'}(\vec{y}')) \leq \varepsilon/3 + 2\varepsilon_1 \leq \varepsilon$, implying that $(m',\vec{y}') \in ReachSet_\varepsilon$. Thus, $B(S_{B',F'},\sigma) \subseteq ReachSet_\varepsilon$.
  Third, since $d(\mathcal{B}) \leq \sigma$, $(m',B') \subseteq B(S_{B',F'},\sigma)$, which implies that $(m',B') \subseteq ReachSet_\varepsilon$.

Thus, due to the above induction method, we completed the proof. □

Now, consider the case that $Jump(m,\vec{x},m',\vec{x}')$ is non-empty. Let

$$Jump_\varepsilon = \{((m,\vec{x}),(m',\vec{x}')) : d((\vec{x},\vec{x}'),(\vec{x}^*,\vec{x}'^*)) \leq \varepsilon, \\ Jump(m,\vec{x}^*,m',\vec{x}'^*)\}.$$

Clearly, $Jump \subset Jump_\varepsilon$. Moreover, $Jump_\varepsilon$ is an ε-perturbed solution set of *Jump*. Similarly, we can define $Jump_{\varepsilon_1}$ and $Jump_{2\varepsilon_1}$.

Due to the definition of $Jump_\varepsilon$, we have the following lemmas.

Lemma 4. *For each pair* $((m,B),(m',B')) \in Trans'_D$, $((m,B),(m',B')) \subseteq Jump_{\varepsilon_1} \subseteq Jump_\varepsilon$. *Moreover,* $((m,B),B((m',B'),\varepsilon_1)) \subseteq Jump_{2\varepsilon_1} \subseteq Jump_\varepsilon$.

Proof. Clearly, $Jump_{\varepsilon_1} \subseteq Jump_{2\varepsilon_1} \subseteq Jump_\varepsilon$. For each pair $((m,B),(m',B')) \in Trans'_D$, we first want to prove: for each $((m,\vec{x}),(m',\vec{x}')) \in ((m,B),(m',B'))$, $((m,\vec{x}),(m',\vec{x}')) \in Jump_{\varepsilon_1}$.

Since $((m,B),(m',B')) \in Trans'_D$, there is a pair $(\vec{x}^*,\vec{x}'^*) \in (B,B')$ such that the constraint $Jump(m,\vec{x}^*,m',\vec{x}'^*)$ holds. Moreover, since $\vec{x},\vec{x}^* \in B$, $\vec{x}',\vec{x}'^* \in B'$, $d(B) \leq \varepsilon_1$, and $d(B') \leq \varepsilon_1$, we have $d(\vec{x},\vec{x}^*) \leq \varepsilon_1$ and $d(\vec{x}',\vec{x}'^*) \leq \varepsilon_1$. Due to the definition of $Jump_{\varepsilon_1}$ $((m,\vec{x}),(m',\vec{x}')) \subseteq Jump_{\varepsilon_1}$. Thus, $((m,B),(m',B')) \subset Jump_{\varepsilon_1}$.

Similarly, we can prove that for each $((m,\vec{x}),(m',\vec{x}')) \in ((m,B),B((m',B'),\varepsilon_1))$, $((m,\vec{x}),(m',\vec{x}')) \in Jump_{2\varepsilon_1}$. Thus, $((m,B),B((m',B'),\varepsilon_1)) \subseteq Jump_{2\varepsilon_1}$. □

Based on Lemma 4, we can easily extend Lemma 3 to the case that the solution set of $Jump(m,\vec{x},m',\vec{x}')$ is not empty.

Lemma 5. *For each* $(m',B') \in \mathcal{B}$, *assume that* (m',B') *is reachable from* $Init'$. *Then for each face* F' *of* (m',B'), $S_{B',F'} \subseteq ReachSet_\varepsilon$ *when* $S_{B',F'} \neq \emptyset$. *Moreover,* $B(S_{B',F'},\sigma) \subseteq ReachSet_\varepsilon$, *implying that* $(m',B') \subseteq ReachSet_\varepsilon$.

Proof. Since (m',B') is reachable from $Init'$, there is a trajectory such that $(m_0,B_0) \in Init'$, $((m_i,B_i)),(m_{i+1},B_{i+1})) \in Trans'$ and $(m_p,B_p) = (m',B')$ for $i = 0,\ldots,p-1$. For proving that for each face F' of (m',B'), $S_{B',F'} \subseteq ReachSet_\varepsilon$ when $S_{B',F'} \neq \emptyset$, we proceed as follows.

- First, assume that there is a certain j such that
  - $((m_j,B_j)),(m_{j+1},B_{j+1})) \in Trans'_D$ (or possibly, $((m_j,B_j)),(m_j,B_j)) \in Trans'_D$), and
  - for all l such that $0 \leq l < j$, $((m_l,B_l),(m_{l+1},B_{l+1})) \in Trans'_C$, $((m_l,B_l),(m_l,B_l)) \notin Trans'_D$ and $((m_l,B_l),(m_{l+1},B_{l+1})) \notin Trans'_D$.

  By Lemma 3, $S_{B_{j-1},B_{j-1}\cap B_j} \subseteq ReachSet_\varepsilon$ and $B(S_{B_{j-1},B_{j-1}\cap B_j},\sigma) \subseteq ReachSet_\varepsilon$. Since $(m_j,B_j) \subseteq B(S_{B_{j-1},B_{j-1}\cap B_j},\sigma)$, $(m_j,B_j) \subseteq ReachSet_\varepsilon$. From Lemma 4,
  - if $((m_j,B_j)),(m_{j+1},B_{j+1})) \in Trans'_D$, then $(m_{j+1},B_{j+1}) \subseteq ReachSet_\varepsilon$ and also $B((m_{j+1},B_{j+1}),\sigma) \subseteq ReachSet_\varepsilon$;
  - if $((m_j,B_j)),(m_j,B_j)) \in Trans'_D$, also $B((m_j,B_j),\sigma) \subseteq ReachSet_\varepsilon$.
- Second, let I be the set

  $$\{i : 0 \leq i \leq p-1, ((m_i,B_i),(m_{i+1},B_{i+1}) \in Trans'_D \text{ or } ((m_i,B_i),(m_i,B_i)) \in Trans'_D\}$$

  and q be the number of elements in I. With the assumption that the elements in I are $l_1,\ldots,l_q$ such that $l_1 \leq \cdots \leq l_q$, we divide the trajectory into $q+1$ parts (i.e.,$r_0,r_1,r_2,\ldots,r_q$) such that
  - for $0 \leq i \leq l_1$, $r_0(i) = r(i)$.
  - for each $k \in \{1,2,\ldots,q-1\}$, if $((m_i,B_i),(m_i,B_i)) \in Trans'_D$, then for each $0 \leq i \leq l_{k+1} - l_k$, $r_k(i) = r(l_k + i)$; if $((m_i,B_i),(m_{i+1},B_{i+1})) \in Trans'_D$, then for each $0 \leq i \leq l_{k+1} - l_k - 1$, $r_k(i) = r(l_k + i + 1)$.
  - if $((m_{l_q},B_{l_q}),(m_{l_q},B_{l_q})) \in Trans'_D$, then for each $0 \leq i \leq p - l_q$, $r_q(i) = r(l_q + i)$; if $((m_{l_q},B_{l_q}),(m_{l_q+1},B_{l_q+1})) \in Trans'_D$, then for each $0 \leq i \leq p - l_q - 1$, $r_k(i) = r(l_k + i + 1)$.

  Due to Lemma 3, $r_0(l_0) \subseteq ReachSet_\varepsilon$. From Lemma 4, $r_1(0) \subseteq ReachSet_\varepsilon$ and $(r_1(0),\sigma) \subseteq ReachSet_\varepsilon$. Again, based on Lemmas 3 and 4, by using the induction method and by proceeding in the same ways as described for proving Lemma 3 and as described in Item (1), we can obtain that for each face F' of (m',B'), $S_{B',F'} \subseteq ReachSet_\varepsilon$ and $B(S_{B',F'},\sigma) \subseteq ReachSet_\varepsilon$ when $S_{B',F'} \neq \emptyset$.

Moreover, since $d(\mathcal{B}) \leq \sigma$, $(m',B') \subseteq B(S_{B',F'},\sigma)$, which implies that $(m',B') \subseteq ReachSet_\varepsilon$. □

Due to the definition of $UnSafe_\varepsilon$ and the similarity to Lemma 2, we have the following lemma.

Lemma 6. *For each $(m,B) \in UnSafe'$, $(m,B) \subseteq UnSafe_{\varepsilon_1} \subseteq UnSafe_\varepsilon$.*

Thus, based on Lemmas 2, 5 and 6, we reach our main result of this paper, which is described as follows:

Theorem 5. *If $\mathcal{H}_\varepsilon$ is safe, then $Abstract_{\mathcal{H}}(\mathcal{B})$ with $d(\mathcal{B}) \leq \sigma$ is also safe, which implies that the abstraction refinement procedure terminates and returns the positive answer.*

Proof. Assume that $Abstract_{\mathcal{H}}(\mathcal{B})$ with $d(\mathcal{B}) \leq \sigma$ is unsafe. We want to deduce a contradiction. Since H_ε is safe, $Init' \cap UnSafe' = \emptyset$ when $d(\mathcal{B}) \leq \sigma$. Thus, there is a trajectory from $Init'$ to $UnSafe'$ such that $r(0) \in Init'$, and $r(p) \in UnSafe'$ and for all $i \in \{1,\ldots,p-1\}$, $r(i) \notin UnSafe'$. Due to Lemma 5, $r(p) \subseteq ReachSet_\varepsilon$. Due to Lemma 6, $r(p) \in UnSafe_{\varepsilon_1} \subseteq UnSafe_\varepsilon$, implying that $\mathcal{H}_\varepsilon$ is unsafe, contradicting with the condition that H_ε is safe. Thus, we proved that if $\mathcal{H}_\varepsilon$ is safe, then $Abstract_{\mathcal{H}}(\mathcal{B})$ with $d(\mathcal{B}) \leq \sigma$ is also safe. Therefore, our abstraction refinement process terminates and returns the positive answer. □

From Theorem 5, we know that for a robustly safe hybrid system, our constraint based abstraction refinement procedure with our remedy will eventually terminate.

5 CONCLUSIONS

Safety verification of hybrid systems is in general undecidable. Due to practical applications, it is sufficient to only consider robustly safe hybrid systems in which a slight perturbation is guaranteed to result in the same desired safety property. In this paper, we provide a constraint based abstraction refinement for safety verification of nonlinear hybrid systems by removing states that do not fulfill the reachability constraint. Moreover, we propose a remedy to reduce the wrapping effect caused by our interval based abstraction refinement. Based on this remedy, we prove that our refinement procedure will terminate for robustly safe nonlinear hybrid systems.

ACKNOWLEDGEMENTS

This work was supported by NSFC-61003021 and Beijing Nova Program. Moreover, the author thanks Dr. Stefan Ratschan for his numerous favorable helps.

REFERENCES

Alur, R. and Courcoubetis, C. and Halbwachs, N. and Henzinger,T. A. and Ho,P.-H. and Nicollin, X. and Olivero, A. and Sifakis, J. and Yovine, S. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138: 3–34.

Damm, W. and Pinto, G. and Ratschan, S. 2007. Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. *International Journal of Foundations of Computer Science (IJFCS)*, 18(1): 63–86.

Fehnker, A. and Ivančić, F. 2004. Benchmarks for hybrid systems verification. In R. Alur and G. J. Pappas, editors, *HSCC'04*, LNCS, Vol. 2993, Springer.

Fränzle, M. 2001. What will be eventually true of polynomial hybrid automata. In N. Kobayashi and B. C. Pierce, editors, *Theoretical Aspects of Computer Software (TACS 2001)*, LNCS, Vol. 2215, Springer-Verlag.

Frehse, G. 2008. Phaver: algorithmic verification of hybrid systems past hytech. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(3): 263–279.

Girard, A. and Pappas, G. 2006. Verification using simulation. In J. Hespanha and A. Tiwari, editors, *HSCC'06*, LNCS, Vol. 3927, pp. 272–286.

Henzinger, T. A. and Kopke, P. W. and Puri, A. and Varaiya, P. 1998. What's decidable about hybrid automata. *Journal of Computer and System Sciences*, 57: 94–124.

Henzinger, T. A. and Raskin, J.-F. 2000. Robust undecidability of timed and hybrid systems. In N. Lynch and B. Krogh, editors, *Proc. HSCC'00*, LNCS, Vol. 1790, Springer.

Julius, A. A. and Fainekos, G. E. and Anand, M. and Lee, I. and Pappas, G. J. 2007. Robust test generation and coverage for hybrid systems. In A. Bemporad, A. Bicchi, and G. Buttazzo, editors, *Hybrid Systems: Computation and Control*, LNCS, Vol. 4416, pp. 329–242, Springer.

Klaedtke, F. and Ratschan, S. and She, Z. 2007. Language-based abstraction refinement for hybrid system verification. In *Proceedings of the Eighth International Conference on Verification, Model Checking and Abstraction Interpretation*, LNCS, Vol. 4349, pp. 151–166, Springer.

Lafferriere, G. and Pappas, G. J. and Yovine, S. 1999. A new class of decidable hybrid systems. In *HSCC*, pp. 137–151.

Neumaier, A. 1993. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. *Computing Supplementum*, 9: 175–190.

Preußig, J. and Kowalewski, S. and Wong-Toi, H. and Henzinger, T. 1998. An algorithm for the approximative analysis of rectangular automata. In *5th Int. School and Symp. on Formal Techniques in Fault Tolerant and Real Time Systems*, LNCS, Vol. 1486, Springer.

Ratschan, S. 2006. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4): 723–748.

Ratschan, S. and She, Z. 2007a. HSOLVER. http://hsolver.sourceforge.net. Software package.

Ratschan, S. and She, Z. 2005. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control*, LNCS, Vol. 3414, pp. 573–589, Springer.

Ratschan, S. and She, Z., 2006. Constraints for continuous reachability in the verification of hybrid systems. In *Proc. 8th Int. Conf. on Artif. Intell. and Symb. Comp., AISC'2006*, LNCS, Vol. 4120, pp. 196–210, Springer.

Ratschan, S. and She, Z., 2007. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions on Embedded Computing Systems*, 6(1).

Ratschan, S. and She, Z., 2008. Recursive and backward reasoning in the verification on hybrid systems. In *Proceedings of the Fifth International Conference on Informatics in Control, Automaton and Robotics*, Vol. 4, INSTICC Press.

She, Z. and Zheng, Z. 2008. Tightened reachability constraints for the verification of linear hybrid systems. *Nonlinear Analysis: Hybrid Systems*, 2(4): 1222–1231.

Van der Schaft, A. J. and Schumacher, J. M. 2000. *An Introduction to Hybrid Dynamical Systems*. Springer.